TCPU CANbus High Level Protocol

Version 1.2 (11.19.04)

Justin Kennington

Communication between the MCU on TCPU and the CANbus host PC comprises:

```
DATA_TO_PC
PLD_SETUP
RESET_TDCS
GET_STATUS
CHANGE_MCU_PROGRAM
DEBUG
```

The first of these, DATA_TO_PC, is a one-way communication initiated by the MCU.

The rest of the transactions are command sequences of two or more messages. PLD_SETUP, RESET_TDCs, GET_STATUS, and DEBUG are initiated by a message the host PC, and completed with a single response from the MCU (containing the requested data, or confirmation of execution of a command). CHANGE_MCU_PROGRAM is more complicated, because more data than the 8 byte limit of each CAN message are required. In this case, an 8 byte data packet is sent (by the PC) and acknowledged (by the MCU) before another 8 byte packet is sent. This continues until all data have been sent.

CANbus Packet ID

Each CANbus packet contains a 29-bit identifier field. Use of the extended identifier format distinguishes CAN messages for TCPU boards from messages for TDIG boards. Packets with a standard (11-bit) ID are ignored. The packet ID specifies the type of message. For run 5 (2004-2005), each TCPU will exist on its own CANbus network (that includes a number of TDIG boards). Therefore, no addressing is necessary. *A TDIG will respond to all extended ID messages it receives.* The data is stored as follows:

```
s-MsgID[10:6] = Message type
s-MsgID[5:0]  = Extra bits (specified per message type if applicable)
e-MsgID[17:0] = Unused
```

Message type

Message type is a 5-bit field that specifies the type of command that the message is related to and whether the packet is a command from the control system (downstream) or response from a TCPU board (upstream). Table 1 shows the available message types and the corresponding codes.

Command	Downstream Code	Upstream Code	Function
DATA_TO_PC	N/A	00000	TDC data upload
PLD_SETUP	00100	00010	Changes operating mode of PLD
RESET_TDCS	01000	00110	Synchronizes all system TDCs
GET_STATUS	01100	01010	Get TCPU status
CHANGE_MCU_PROGRAM	10000	01110	Change MCU firmware
DEBUG	10100	10010	Multifunction debugging code
MASTER/SLAVE	11000	10110	Sets the TCPU as master or slave
ERROR	N/A	11110	Response to unrecognized message

Table 1

DATA_TO_PC

This command is used to push data upstream from the MCU. These messages only flow one direction, and the MCU assumes that the messages are received.

```
MsgID = 0x00000000
```

Payload:

For bench testing and debugging, data can be sent via the CANbus. Each data word is four bytes, so the MCU will typically pack two words per CAN message. However, if only one word is available, the MCU will send a CAN message with a single word, to avoid stalls. The resulting two payload formats are:

```
dddddddd dddddddd dddddddd dddddddd
       [four byte data word]

dddddddd dddddddd dddddddd dddddddd   dddddddd dddddddd dddddddd dddddddd
     [1st four byte data word]              [2nd four byte data word]
```

Information identifying the data type is included in the data word itself. The exact source of this data and whether data is sent to the CANbus at all is determined by registers set in the PLD, under CANbus control (see “PLD SETUP”).

PLD_SETUP

This command is used to configure the three registers inside TCPU's PLD used for setting different data transfer modes, *etc*. There are three registers in the PLD: Mode, Configuration, and MCU filter.

For the PC Command:

```
SID[10:6] = 00100
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x04000000
```

For the MCU response:

```
SID[10:6] = 00010
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x02000000
```

Payload

PC to MCU (Command):

The payload of a PLD_SETUP command includes two bytes. The first is an address from 0x0 to 0x7. The second byte is the byte to be stored at the register address. The function of each register and the various settings may be found in other TCPU documentation. The payload format looks like this:

```
00000aaa bbbbbbbb

a = register address (0x0 – 0x7)
b = byte to store
```

MCU to PC (Response):

The response payload for TDC status requests *resembles* an echo of the command payload. Instead of blindly echoing the received command, however, the MCU attempts to read back the data previously written. The read-back data is stuffed into the CANbus response message. Note that some PLD registers are non-readable. Writing to these registers may result in non-matching CANbus replies. See TCPU documentation for details.

RESET_TDCS

This command is used to reset the bunch reset counters of all TDCs on the network (start and stop) to synchronize them and ready them for data acquisition.

For the PC Command:

```
SID[10:6] = 01000
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x08000000
```

For the MCU response:

```
SID[10:6] = 00110
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x06000000
```

Payload

PC to MCU (Command):

The payload of a TDC reset command contains no data. It is zero bytes long.

MCU to PC (Response):

The response to a TDC reset command is also a zero-byte payload. Receipt of this message indicates that the bunch reset pulse has been sent to all TDIG boards in system. **This command should only be sent to the master TCPU**, which will send a synchronized pulse to all slaves.

GET_STATUS

This function has a command and response structure. The command tells the MCU to retrieve the status byte from the PLD and send it to the host PC. The meaning of the status word can be found in other TCPU documentation.

For the PC Command:

```
SID[10:6] = 01100
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x0C000000
```

For the MCU response:

```
SID[10:6] = 01010
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x0A000000
```

Payload

PC to MCU (Command):

There is no payload for a status request command. It is a zero-length message.

MCU to PC (Response):

The response payload for TDC status requests is one byte and is an exact copy of the byte read from the PLD status register. The response payload will be formatted as follows:

```
ssssssss

s = TCPU PLD status[7:0]
```

DEBUG

Instruction space is reserved for a further command/response pair for debugging. The commands will consist of requests for various data and/or calls to various diagnostic functions. These functions will be used to allow the CAN bus to detect and/or correct system errors. Further definition of these functions will be added later.

For commands issued by the PC to the MCU:

```
SID[10:6] = 10100
SID[5:0]  = Unused (don't care)
EID[17:0] = Unused

MsgID = 0x14000000
```

For responses by the MCU to the PC:

```
SID[10:6] = 10010
SID[5:2]  = Unused (don't care)
EID[17:0] = Board number

MsgID = 0x12000000
```

MCU Restart

The DEBUG command can be used to restart the MCU. This is as close to a power-off reset as one can get without actually turning off the power.

Header

Bits [5:0] of the header are ignored and may be set to any value. They will be echoed in the response.

Payload

The payload is an arbitrary 8 digit code to prevent accidental restarts:

```
0100 0101 0110 1001 0011 0011 0001 0100
HEX: 0x45 69 33 14
```

PLD Reset

This command is used to reset the PLD's internal state machines and flush data FIFOs.

Header

Bits [5:3] of the header are ignored and may be set to any value. They will be echoed in the response.

Payload

Like MCU reset, the payload consists of an arbitrary four byte code:

```
0110 1001 1001 0110 1010 0101 0101 1010
HEX: 0x69 96 A5 5A
```

MASTER/SLAVE

This function is used to instruct a TCPU board to function as a master or slave. Master TCPUs run from their own on-board oscillators and provide this oscillator to the slave TCPUs. Slaves run from the clock provided by a master.

For the PC Command:

```
SID[10:6] = 11000
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x18000000
```

For the MCU response:

```
SID[10:6] = 10110
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x16000000
```

Payload

PC to MCU (Command):

The payload is one byte.

```
0xAA = Function as master
0x55 = Functino as slave
```

MCU to PC (Response):

The response payload from TCPU will be a copy of the command payload.

ERROR

When the MCU receives a CAN message that it does not understand, it replies with an error packet. Error conditions include: unrecognized message ID and malformed payload. Note that a configure_TDC or reprogram_MCU command received in error will result in a configure_TDC or reprogram_MCU response packet with its error bit set.

```
MsgID[10:6] = 11110
MsgID[5:0]  = Echo
```

CHANGE_MCU_PROGRAM

This command sequence is used to change the program code running on the MCU or PLD. The CHANGE_MCU_PROGRAM sequence is similar to the CONFIGURE_TDC sequence, but with added safety features. The format used for the MsgID is slightly altered to make the transaction more efficient.

For commands issued by the PC to the MCU:

```
SID[10:6] = 10000
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x10000000
```

For responses by the MCU to the PC:

```
SID[10:6] = 01110
SID[5:0]  = Unused
EID[17:0] = Unused

MsgID = 0x0E000000
```

Overview:

The MCU (PIC18LF8720) has 128 kilobytes of internal program memory space, which is easily more than double the amount of program code used in TDIG (currently less than 8 kilobytes). As initially programmed, the program code will reside in the lower half of program memory, beginning at address 0x00000. When updated firmware is applied, the new code will be written to the upper half of program memory (starting at address 0x10000), and once written, execution will jump to that space. Upon reboot, a CAN message will need to be sent to jump execution back to the reprogrammed code.

Data will be programmed in groups of 64 bytes. Every 64 bytes the data will be checksummed, and a Program_64 command will be issued. Once the checksum is verified by the MCU, CAN communication with the MCU will pause while the MCU writes the 64 bytes to flash memory. After writing and verifying 64 bytes, the MCU will confirm that they have been written. This process will continue until all program changes are complete. At that time, a final checksum will be performed for the whole program code. Once verified, the PC will issue a JUMP_PC command. The MCU will acknowledge the JUMP_PC command and jump its program counter to the start of the new code. It is advised to include code at the beginning of the new code that will send a CAN message confirming execution of the new code.

To change the code a second time, the MCU will need to be rebooted to the original code, and the previous updated code will be overwritten. This way, the original code will never be overwritten. It is impossible for this document to promise any features in the newly programmed code. However, **it is strongly advised that the new code not be capable of reprogramming *any* flash memory.**

Payload:

There are several different types of packets involved in MCU reprogramming. Each type has a particular function and format. The type is identified by the sub-instruction field, stored in the descriptor that forms the first byte of all reprogramming packet payloads.

Descriptor:

The first byte of all reprogramming payloads is a descriptor field that gives information about the command type and any errors that have occurred. The descriptor field is laid out as follows:

```
DES[7]   = error
DES[6]   = error (where used)
DES[6:4] = not used (don't care)
DES[3:0] = sub_instruction
```

The sub_instruction field is used to determine which of the reprogramming packet types commands the packet represents.

Error bits:

An error bit is set by the MCU to indicate that an error has occurred. If the MCU detects an error, it will **perform no further action**, and will send back the message payload with the appropriate error bit set. The meaning of the error bits is different for each sub_instruction, and is described below. If the PC receives a packet with the error bit set, the safe response is to restart the current 64 byte block by sending a new start sub_instruction and re-sending all data.

MCU to PC packets (Responses):

The MCU has a response to each packet sent by the PC, and the PC should wait for the appropriate response to a packet before sending the next packet. The MCU responds to PC packets by changing the Message ID. That is,

```
MsgID[10:6] = 01110
MsgID[5:0]  = Echo (same as sent by PC)
```

Each error-free response by the MCU indicates that a particular action has been taken, dependent on the sub_instruction. The actions taken by the MCU are described below.

Start Packet:

```
DES[3:0] = 0000
```

Format:

The start command's payload contains only the descriptor.

```
e000 0000

e = error bit
```

Effect:

The start command instructs the MCU to ready itself for the transfer of 64 bytes of new program data. Data will follow in later packets. Sending the start sub_instruction will always re-start the 64 byte transfer. Error-free response indicates that the MCU has received the start command and is prepared to begin receiving new program data.

A start command may be issued at any time, and will always any current reprogramming sequence.

Errors:

An error bit will be set if the MCU is not ready to accept new program data.

Data Packet:

Format:

```
DES[3:0] = 0001 (data)
```

Actual program data is sent in data packets, each of which contains four bytes of new program data. In addition to the descriptor and program data, each data packet contains a 16-bit start address. The start address is the address that the first data word is written to. The next data word is written to start address + 1. The first data packet in a 64 byte block must have a start address of a 64 byte block. Therefore the 6 LSB's will be zero (*e.g.*, 0x000, 0x040, 0x080, *etc*.).

```
e000 0001 aaaaaaaa aaaaaaaa 1ddddddd 2ddddddd 3ddddddd 4ddddddd

e = error bit
a = start address (MSB first)
1d = first data word
2d = second data word
```

Effect:

When the MCU receives a data packet, it will write the data words to data memory. Error-free response indicates that the MCU has received the new program data, written it to data memory, and is ready to receive following information.

The PIC18's 128 kilobyte address space is addressed with 17 bits. Only 16 bits of address are sent, and a leading 1 is appended by the MCU before writing data. This assures that only the top half of memory can be written with new code. Sending data with a start address of 0x0000 will result in that data being written to program memory at 0x10000. Of course, safety mechanisms like this **cannot** be guaranteed in the new code.

Errors:

A data packet must be preceded by a start instruction or another data packet. The error bit is set under either of two conditions:

1. The preceding packet was a start instruction, but the data packet address is not the start of a 64-byte sector. A first data packet will always have a start address that is the beginning of a 64 byte sector (that is, the 6 LSB's will be zero).
2. A non-first data packet has a start address not equal to the previous data packet start address plus four.

Checksum_64 Packet:

Format:

```
DES[3:0] = 0010 (chksum_64)
```

The message payload contains the descriptor, the start address of the just-transferred 64-byte block, and the 8-bit checksum of that 64-byte block.

```
e000 0010 aaaaaaaa aaaaaaaa cccccccc

e = error bit
a = 16-bit address for START of new program data
c = checksum of previous 64 program bytes
```

Effect:

The Checksum_64 command is used to confirm the successful transmission of one 64-byte block. It checks the program code that is temporarily stored in data memory, *before* the code has been written to flash program memory. Therefore **this message is only valid following a completed 64-byte data transaction**. It is also ok to send this command following a Program_64 programming error.

When the MCU receives this command, it calculates a checksum from the 64-byte temporary program array in data memory. It then compares the calculated value to that transmitted in the CAN message. If and only if there is a match, the MCU will allow a following Program_64 command (see below) to execute.

Errors:

Three conditions can cause an error in response to a Checksum_64 packet.

1. The preceding packet was not a Data packet or a Checksum_64 packet
2. The start address provided does not match the expected start address.
3. A checksum mismatch occurs.

In all cases, the error bit will be set. In the case of a checksum mismatch, the calculated checksum will be appended to the returned message, like so:

```
1000 0010 aaaaaaaa aaaaaaaa cccccccc xxxxxxxx

e = error bit (will always be 1 in this case)
a = 16-bit address for START of new program data
c = checksum sent by PC
x = data memory checksum calculated by MCU (mismatch)
```

A packet error (cases 1 and 2) response will not include the calculated checksum. Therefore, it is possible to determine which type of error occurred by checking the length of the returned error packet.

Program_64 Packet:

Format:

```
DES[3:0] = 0011
```

The payload contains the descriptor and a start address.

```
ex00 0010 aaaaaaaa aaaaaaaa

e = address error bit
x = programming error bit
a = 16-bit address for START of new program data
```

Effect:

The Program_64 packet is the final transaction for writing a 64 byte block to program memory. This packet is only accepted if the previous transaction was a successful Checksum_64. When the MCU receives a Program_64 packet following a successful checksum, the 64 bytes of new program code are written to FLASH program memory beginning at the start address included in the Program_64 packet and continuing through this address + 63. After the data has been written to program memory, it will be checked against the copy stored in data memory. An error-free response to this packet indicates that the previous 64 bytes have been successfully written to program memory and verified.

Errors:

The Program_64 response has two error bits. The address error bit is set if the preceding packet was not a Checksum_64, or if the start address provided does not match the expected start address. The start address must match that of the preceding Checksum_64 packet.

The programming error bit is set if there is a mismatch between the program data temporarily stored in data memory (and previously checksum-verified) and the data programmed into Flash program memory. If a programming error is detected, it is ok to resend the corresponding Checksum_64 packet and re-try the Program_64.

Final_chksum Packet:

Format:

```
DES[3:0] = 0100
```

The payload of a Final_chksum packet includes the descriptor, a final address, and an 8-bit checksum that is the 8-bit sum of all program memory data from 0x0000 to the final address. This is a checksum for all of the new program code.

```
e000 0011 aaaaaaaa aaaaaaaa cccccccc

e = packet error bit
x = checksum error bit
a = program code end address. Checksum is calculated from 0x0000 to this
     address (but not including this address!)
c = 8-bit checksum of new program code
```

Note that the address sent with the Final_chksum packet is *outside* the address space used by the new program code. It is equal to the highest address of the final 64-byte sector *plus one*. Therefore this address will be a 64-byte aligned address (6 LSB's = 0).

Effect:

This packet is a final checksum for the entire new program code stored in FLASH program memory. If this checksum fails, there is little that can be done to easily correct the problem. The entire programming sequence must likely be restarted. However, it is a necessary final check to ensure that the new program code has made it uncorrupted into program memory. An error-free response to this packet indicates that the entire program code has successfully made it into program memory and is ready for execution.

Errors:

Three conditions can cause an error in response to a Final_chksum packet:

1. The Final_chksum packet was not preceded by a Start or Program_64 packet.
2. The Final_chksum packet follows a Program_64 packet, but the program code end address does not match the expected end address (calculated by adding 0x40 to the Program_64 address).
3. A checksum mismatch occurs.

In all cases, the error bit will be set. In the case of a checksum mismatch, the calculated checksum will be appended to the returned message, like so:

```
1000 0010 aaaaaaaa aaaaaaaa cccccccc xxxxxxxx

e = error bit (will always be 1 in this case)
a = 16-bit address for START of new program data
c = checksum sent by PC
x = data memory checksum calculated by MCU (mismatch)
```

An address or packet mismatch (cases 1 and 2) response will not include the calculated checksum. Therefore, it is possible to determine which type of error occurred by checking the length of the returned error packet.

Jump_PC Packet

Format:

```
DES[3:0] = 0101
```

The payload of a Jump_PC packet includes only the descriptor.

```
e000 0100

e = error bit
```

Effect:

The Jump_PC packet instructs the MCU to jump execution to address 0x10000, the start location of updated program code. *This packet must be preceded by a successful Final_Chksum packet.* There is no guaranteed response to this message, as it causes the program counter to begin executing new code. It is recommended that the new code be programmed to send a confirmation upon startup.

Errors:

The error bit is set if a Jump_PC packet is preceded by anything other than a successful Final_Chksum packet.

Summary

The key to understanding the reprogramming sequence is to understand what each command does, and the circumstances under which each command is allowed to execute. Some commands have more than one allowable preceding command. This chart will help:

Command	Required Preceding Command 1	Required Current Address 1	Required Preceding Command 2	Required Current Address 2
Start	Any	Any		
Data	Start	'xxxxxxxx xx000000'	Data	Previous + 0x4
Checksum_64	Data	Previous + 0x3C	Checksum_64	Previous
Program_64	Checksum_64	Previous		
Final_chksum	Start	Any	Program_64	Previous + 0x40
Jump_PC	Final_chksum	N/A		

Important notes:

Following an MCU restart, the MCU will be executing old code, not the reprogrammed code. In order to switch to the new code, send a start command, followed by a Final_chksum (yes this means the checksum *must* be known!), followed by a Jump_PC command. One side effect of allowing this sequence is that a Final_chksum packet is not required to follow a Program_64 packet. However, it is **strongly advised** that when new program data is initially written, you follow the final Program_64 packet with a Final_chksum packet. This helps to ensure that the entire memory space that was supposed to be written has actually been written.

MCU Startup Message

In order to keep the user of a potentially reprogrammed system informed, a message is sent at the beginning of the MCU code. This message uses the ID of a reprogramming packet, and a message of this type should be used in newly reprogrammed code to indicate successful execution of upper memory code.

Format:

The payload of a startup packet includes an upper byte of all 1's followed by three bytes of all zeros, representing the memory location of the startup code (location 0x00). Newly reprogrammed code should indicate its memory location by inserting 0x10000, which is the startup location of the new code.

```
11111111 00000000 00000000 00000000
Hex: 0xFF 00 00 00
```